**Git Flow –** **1)** Create branch **2)** Add commits **3)** Open a pull request **4)** Discuss & Review code **5)** Merge **6)** Delete your branch

**Merge Strategies –** Git reset will change the commit ID. ***Fast Forward:*** History is one straight line with no new commits on master. ***Recursive:*** New commits on master & other branches

**SDK –** Set of libraries & tools for developing & running .NET applications. Like Java JDK. When you download SDK, you automatically get the runtimes with it. Install on machines to use for development.

**Runtime –** Set of libraries & tools for running .NET applications. Runtimes provides primitive data types & fundamentals utilities. Install on machines that you want to prepare for running .NET apps.

**SELF-CONTAINED deployment model –** Does NOT need .NET runtime

**Framework-Dependant deployment model –** Needs .NET runtime

**“Using” -** Can only be used before or right after the name space. 3 uses, look at notes: *orange*

**Value Types –** Can’t be inherited (Sealed). Doesn’t contain heap memory address. Always has a value (Won’t be null –> 0) Derived from System.ValueType. Structs & Enums are Value types. Can only represent null using ? int? X = null;

**Reference Types –** Supports inheritance. Contains memory in heap. Always contains NULL when not initialized. Classes & Record are Reference types String c = s;

**Implicit Types: -** Can create a local variable along with initialization without explicit type declaration using var. var does not mean variant and can’t indicate if variable is loosely typed. Used for convenience. var i = 5; -> int i = 5;

**Anonymous Types –** Convenient way to encapsulate a set of read-only properties into a single object without having to explicitly define a type first. var v =new {amount = 108, message = “Hello”};

**Implicit Casting –** Occurs Automatically, guaranteed to succeed, no info loss. Look at notes: *orange*

**Explicit Casting –** Requires cast, may not succeed, Info may be lost. Look at notes: *orange*

**Structs –** Primary purpose is to store data values, can’t declare a base type, they derive from System.ValueType. Implicitly sealed.

**Interfaces –** Can contain *methods, properties, events, and indexers*. Interfaces may employ multiple inheritance. Name usually starts with a /.

**Enums –** Defines a type for a related group of symbolic constants. Choices must be known at compile time. ***Strongly typed:*** No implicit conversion to/from int. Can be explicitly converted. Operators: +,-,++,--,&,| ,~,^. ***Can specify underlying type:*** byte, sbyte, short,ushort,int, uint,ulong

**Properties –** A natural extension of fields. Does not take storage location. Properties have accessors that specify statement executed when their values are read or written.

**Indexers –** Similar to properties. Allows instances of a class or struct to be indexed just like arrays.

**LINQ –** **L**anguage **In**tegrated **Q**uery. SQL like statement designed to allow you to query data structures. Works with generic collections, databases, and xml documents. Used Query syntax.

**Lambda Expressions –** Simple syntax to create in line functions. Used heavily in conjunctions with LINQ and anonymous methods.

**Static Content –** Often user created, unchanged without manual intervention, GeoCities- style web – dynamic website.

**Dynamic Content –** Data driven (stores websites), User driven (Facebook, snapchat)

**Active Server Page(ASP) & Personal Homepage (PHP) -** Look at notes: *orange & yellow*

**Model View Control (MVC) –** ***Model:*** Logic of the application, follows the business rules. ***View:*** Displays the user interface. ***Controller:*** Handles user interaction. Works with the model & the view. Strongly typed views give view access to the model. Description of each look at notes: *orange & yellow*

**Strongly Typed Views -** Bound to the model. The preferred way of transferring data from controller to the view. VSC will help. In order to create a strongly typed view in ASP.NET, we need to pass the model object as a parameter to the View() extension method. The Controller base class provide us the following four overloaded versions of View() extension method which we can use to pass the model data from the controller action method to a view. Strongly typed view gives view access to the model. Can be mixed with other views.

**Weakly Typed Views -** Not Bound to a model. Controller passes data to view using session data or databag/viewData. VSC will not help. Is the preferred way of transferring small data from controller to view. Can be mixed with other views.

**Dynamic Views -** Not bound to a model. Data is still passed to the view in model object. VSC will not help. Least preferred way of transmission. You may use dynamic views for transferring big object data from Controller to View but not defining what the object structure in advance. Can be mixed with other views.

**Partial Views -** A razor markup file (.cshtml) that renders HTML output with another markup files rendered output. Breaks up large view files into smaller components. Reduces duplication of common markup content across view files.

**HTML Forms –** A data collection device <forms></forms>

**ASP.NET Forms –** Identical to HTML Forms. NOT identical to ASP.NET Webforms

**Requests –** Made for something originating a URL, data collected from forms to pass to the server. Website, Image File

**Response –** Sent back from the og requests. Response contains the raw data that describes the request. HTML files and JPEG.

**ASP.NET MVC Core request pipeline –** Look at notes: *yellow*

**Middleware –** Software that’s assembled into an app pipeline to handle requests and responses. Its executed on every request. App.Use(): define a chain middleware (can be terminal) app.Map(): define a branching middleware(can be terminal) app.Run(): define a terminal middleware

**Routing -** Term for navigating around your application. Responsible for matching incoming HTTP requests & dispatching to the endpoints. Why? -> ASP.NET, MVC & MVC CORE applications no longer use static files & locations to display their content. Each route points to a predefined, precompiled location & not necessarily a file. Apps can configure routing using controllers, Razor pages, signal R, GRPC Services, Endpoint enables middleware (such as health checks) and Delegates & Lambdas. ***Reserved routing words:*** Action, Area, Controller, Handler, Page

**Endpoint -** The applications units of executable request-handling code. Selected by matching URL & HTTP methods. Executed by running the delegate.

**URL Matching, Process, Routing Using Controllers, Route Template, Conventional Routing and REST Attribute Routing -** Look at notes: *yellow*

**Binding –** The ASP.NET Core gives us several attributes to control and choose from what source we want to receive the binding data. [FromForm][FromRoute][FromQuery][FromBody][FromHeader][FromServices]

**Entity Framework (EF) –** Relational-Database-Mapping software built for .NET. Provides programmers with easy-to-use interface to a database. Data access preformed using the model. A model is made of entity classes & a context object. ***Entities:*** Tables, Views, Stored procedure. ***Data Context:*** Programmer interface. Code-less CRUD web interfaces.

**Object-Relational Mapper (ORM) –** Used to create a “code version” of your database & vice versa. ***Tables*** are turned into model classes called entities. ***Entities*** are tables, views, stored procedure, and index. ORM creates a programming interface called data context. ORM & EF create a separate Data Access layer.

**Program structures –** ***Assemblies:*** .exe or .dll -> Namespace -> ***Programs:*** declares types, inside namespace -> ***Types:*** Classes, structs & interfaces -> ***Members:*** Fields, methods, properties & events.

**DBContext –** Materialized data returned from the database as entity object. Track changes that were made to the objects. Handles concurrency. Propagate object changes back to the database. Binds objects to controls. The primary class that is responsible for interacting with data as objects. Maps to a specific database that has a schema that DBContext understands. Accessed through lCollections<>

**DBSet –** DbContext represents the collection of all entities in the context by the help of DBSets. Including a DBSet type to your context meants that it is included in EF core model. DBContexr Class includes a DBSet for each entity of your app. Without DBSet the connection between DBContext and the database is not created.

**VCS, Relational Database (SQL Server/Azure SQL) Async Programming -** Look at notes: *Green*

**Async, await & task –** You need to define the method to be async to use await in it. Async methods usually return a task***.*** Breakfast example. ***Naming Convention:*** For async methods returning task add sync at the end of the method name. For async methods returning void, add Begin or Start to the beginning of the method name.

**File upload methods – *Buffered model binding for smaller files****:* The entire file is read into an lFormFile(might cause site to crash if file is to large if app has to many updates) ***Unbuffered Streaming for larger files:*** Reduces risk of site crash. Does not improve performance. The file is received from a multipart request and directly processed or saved by the app.

**Data Storage Types - *Database:*** Less expensive, for small files, faster and easier. ***Cloud Storage:*** Better scalability, better reliability, better security. ***Web Server:*** Less expensive than Cloud Storage. For large files, App must have read/write access.

**Azure Storage -** Provides flexible options to store data in the cloud. Storage mechanism for database applications.

**Azure Blob Storage –** Used to optimize storing massive amounts of unstructured data, such as text or binary data. Objects in blob can be accessed from *anywhere in the world via http or https*. Ideal for images, videos or documents in a browser. Advantages of blob is that its low cost, high availability, strong consistency and disaster recovery capability.

**Model State –** You need to proactively check for validity of the model. Model state object provides details about the state of your model. Updated by model binding process.

**Model Validation –** Both model binding & model validation occurs before the execution of a controller action. Web apps typically redisplay the page with an error message.

**Client Side Programming –** Typically JS applications that are downloaded to the client browser & the JS takes control of drawing the page content. The content is pulled from a web service. jQuery, AngularJS, reactJS, KnockoutJS <- (JS Technologies) (CSS Technologies) -> Bootstrap, CSS

**MVVM –** Model, View and ViewModel. ***ViewModel:*** Accountable for presenting functions, methods, and commands to uphold the state of the view, operate the model and activate the events in the view itself.

**Blazor and Razor Components -** Look at notes: *Green* Look at picture below.

**3 Characteristics for a file to act like a Razor page - 1)** Can’t have a leading underscore in its file name. **2)** The extension is. cshtml **3)** First line in the file is @page (Must have or it wont be routed)

**What is a Web Service? What is REST? -** Look at notes: *Green*

**REST Components –** Clients, API, Server. Look at notes for more details about each: *Green*

**REST Principals –** Client-Server, Stateless, Cacheable, Uniform Interface, Layered System. Look at notes for more details about each: *Green*

**Controller Class –** Base class for an MVC controller with View support. Extends controllerBase class. Mainly handles web pages. Public class MyviewsController : Controller

**ControllerBase Class –** Base class for an MVC controller without view support. The ultimate base class for all controllers in MVC core. Handles web APIs. Does not support web pages.

**Swagger (Open API) –** OpenAPI is a language agnostic specification for describing REST APIs. Swagger is one of the most popular REST API documentation tool. It allows both humans and computers to understand the capabilities of a REST API without direct access to the source code.

**Logging -** Look at notes: *Green*

**Areas -** Look at notes: *Green*

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Composit Primary Key:** modelBuilder.Entity<Client>() .HasKey( x => new { x.UserID, x.ClientID } );

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**How to enable routing in program.cs:**

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapRazorPages();

});

//app.MapGet("/", () => "Hello World!");

app.Run();

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Write a LINQ/Lambda expression to query a collection • Write code to query a database to get specific data to load for a View

using System.Data.Linq;

// Assume that the database is named "MyDatabase" and has a table named "Customers"

DataContext db = new DataContext("MyDatabase"); // Query the Customers table to retrieve all customers with the last name "Smith" var customers = from c in db.Customers

where c.LastName == "Smith"

select c; // Pass the results of the query to the View return View(customers);

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

CODE EXAMPLES ASSIGNMENT 1:

* LINQ & LAMBDA EXPRESSIONS
* Layout page & Razor Basics
* Form Tag & Tag helpers
* Conventional Routing & Declaration of multiple Routes
* Attribute routing (URL)
* Model Binding
* DBContext, DBSet & Queries
* Entity keys, Attribute/Data Annotations & Generated Values
* Async method modifier, Await modifier & Task Class
* Action Return Types

** **

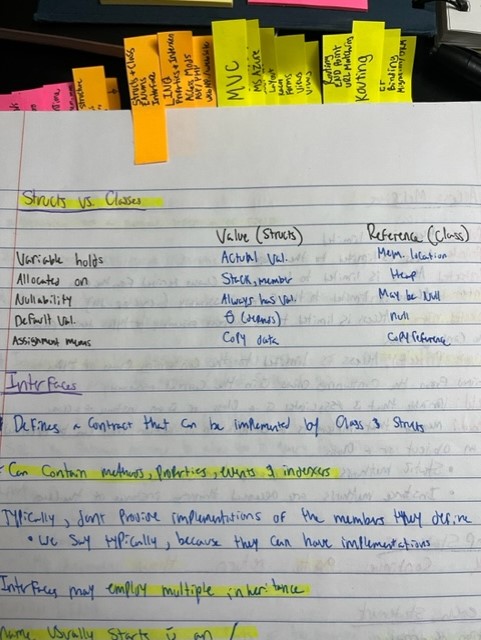
**Weakly typed view Strongly typed view**

* C# can be used to create projects that will look and act like standard Windows programs.  **TRUE**
* When you begin to write the C# code for a project that has been carefully planned, you can create powerful programs with relatively few lines of C# code. **TRUE**
* The C# language is not case sensitive. **FALSE**
* You should always set the Name property of your controls before writing code so the method names will match the control names. **TRUE**
* Each class of controls has the same set of properties. **FALSE**
* More than one control on a form can have the focus at the same time. **FALSE**
* If you break C#'s rules for punctuation, format, or spelling, you will generate a syntax error. **TRUE**
* C# gives the programmer the ability to set up locations in memory and give each location a name. **TRUE**
* In the statement this.Close(); the keyword this refers to **the current object**
* What file extension is given to the name of the solution file for a C# application? **.SLN**
* Comment statements begin with **//**
* Methods always have **parenthesis**
* Most C# statements must be terminated by a **semicolon**
* A new window added to a C# application is called a **FORM**
* What’s a website? **Series of XML-like files that uses a language called HTML**
* What’s a web application? **Languages like ASP or PHP that connects with the server**

**Diagram

Description automatically generated**

Figure -MVC



Structs vs. Classes

Text, letter

Description automatically generated

Blazor and Razor Components